ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

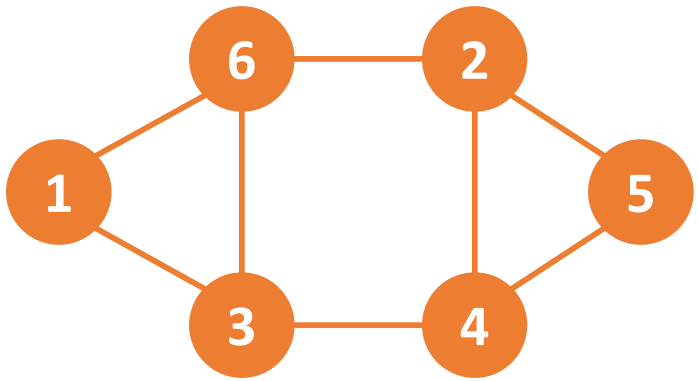
Đồ thị

NỘI DUNG

- Khái niệm và định nghĩa
- Biểu diễn đồ thị
- Duyệt đồ thị
- Đồ thị Euler và đồ thị Hamilton
- Cây khung nhỏ nhất của đồ thị
- Đường đi ngắn nhất trên đồ thị

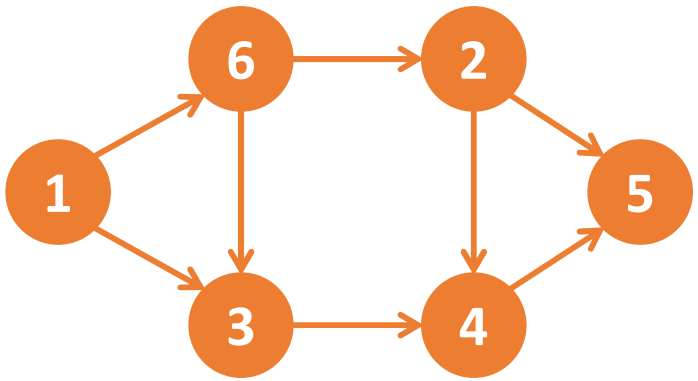
KHÁI NIỆM VÀ ĐỊNH NGHĨA

- Đồ thị là một đối tượng toán học mô hình hoá các thực thể và các liên kết giữa các thực thể đó
- Đồ thị $G = (V, E)$ trong đó V là tập đỉnh và E là tập cạnh (cung)
- Với mỗi $(u, v) \in E$: ta nói v kề với u



Đồ thị vô hướng

- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1, 3), (1,6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 6), (4, 5)\}$



Đồ thị có hướng

- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1, 3), (1,6), (2, 4), (2, 5), (6, 2), (3, 4), (6, 3), (4, 5)\}$

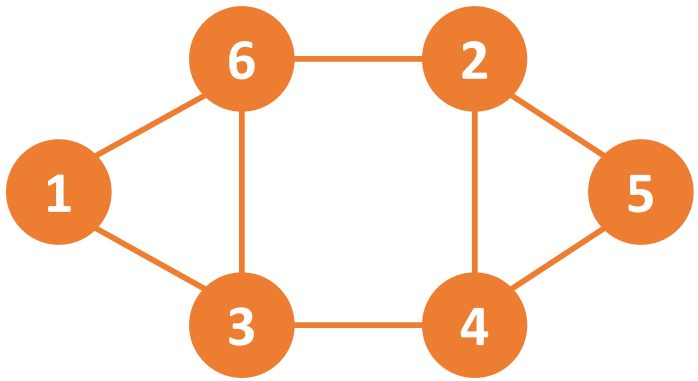
KHÁI NIỆM VÀ ĐỊNH NGHĨA

- Bậc của một đỉnh trên đồ thị vô hướng là số đỉnh kề với đỉnh đó:

$$deg(v) = \#\{u \mid (u, v) \in E\}$$

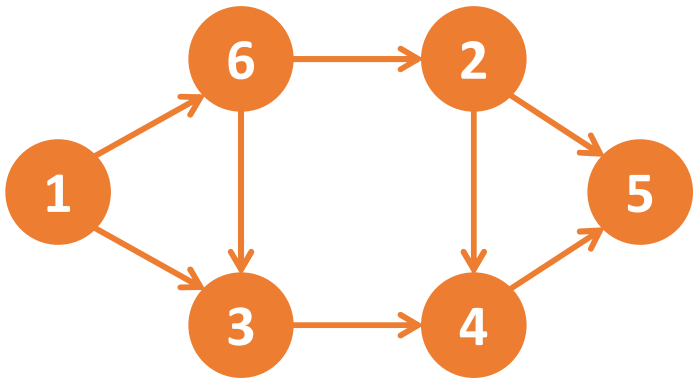
- Bán bậc vào (ra) của một đỉnh trên đồ thị có hướng là số cung đi vào (ra) đỉnh đó:

$$deg^-(v) = \#\{u \mid (u, v) \in E\},\ deg^+(v) = \#\{u \mid (v, u) \in E\}$$



đồ thị vô hướng

$deg(1) = 2$, $deg(4) = 3$

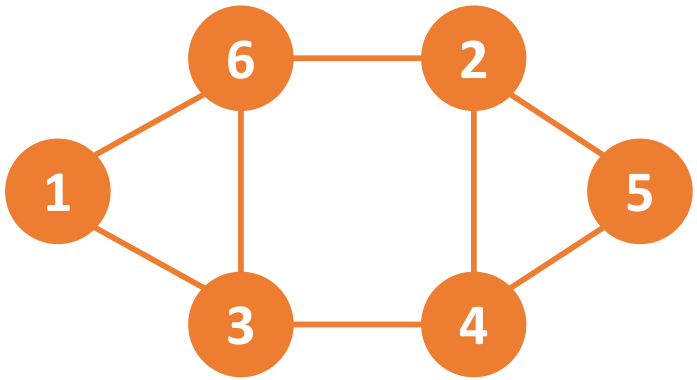


đồ thị có hướng

$deg^-(1) = 0$, $deg^+(1)=2$

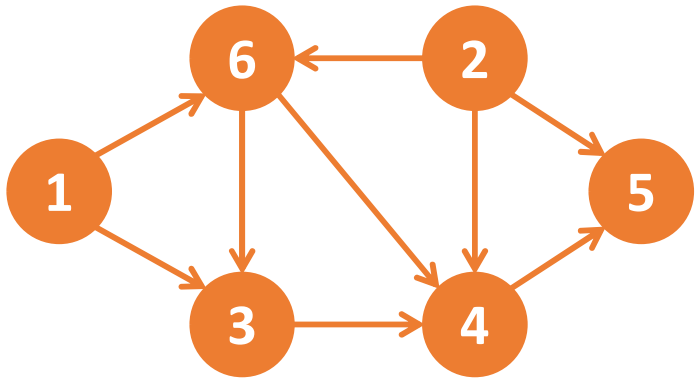
KHÁI NIỆM VÀ ĐỊNH NGHĨA

- Cho đồ thị $G=(V, E)$ và 2 đỉnh $s, t \in V$, đường đi từ s đến t trên G là dãy $s = x_0, x_1, \ldots, x_k = t$ trong đó $(x_i, x_{i+1}) \in E$, với $\forall\ i = 0, 1, \ldots, k\text{-}1$



Đường đi từ 1 đến 5:

- 1, 3, 4, 5
- 1, 6, 2, 5

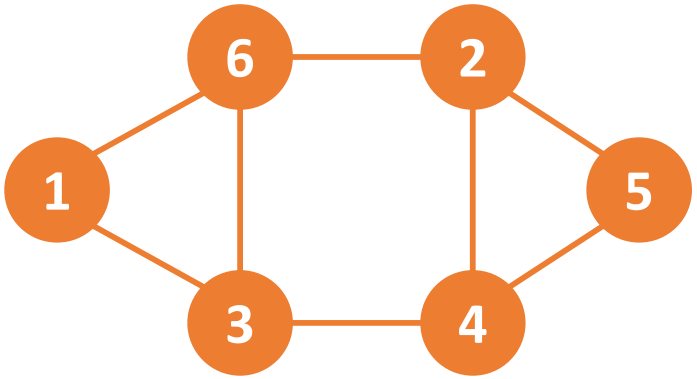


Đường đi từ 1 đến 5:

- 1, 3, 4, 5
- 1, 6, 4, 5

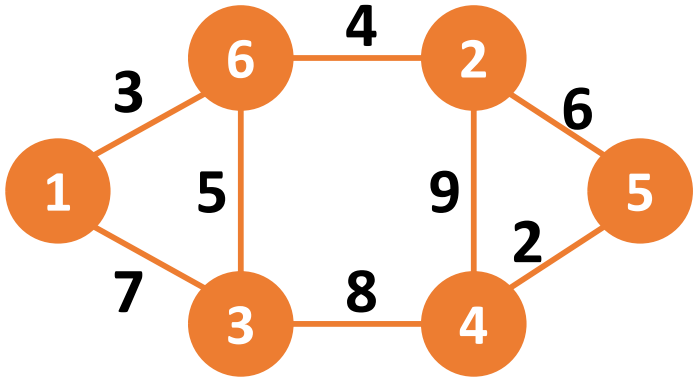
BIỂU DIỄN ĐỒ THỊ

- Ma trận kề



	1	2	3	4	5	6
1	0	0	1	0	0	1
2	0	0	0	1	1	1
3	1	0	0	1	0	1
4	0	1	1	0	1	0
5	0	1	0	1	0	0
6	1	1	1	0	0	0

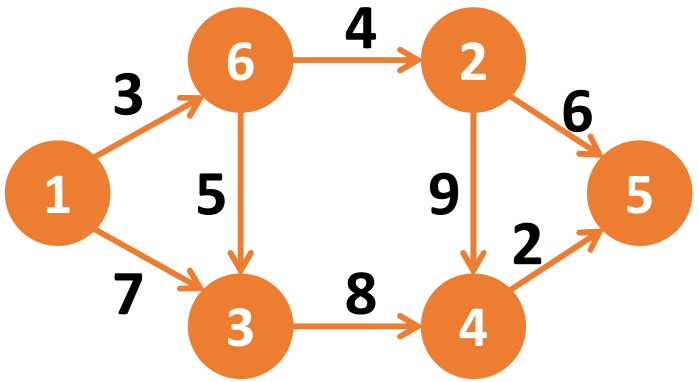
- Ma trận trọng số



	1	2	3	4	5	6
1	0	0	7	0	0	3
2	0	0	0	9	6	4
3	7	0	0	8	0	5
4	0	9	8	0	2	0
5	0	6	0	4	0	0
6	3	4	5	0	0	0

BIỂU DIỄN ĐỒ THỊ

- Danh sách kề
  - Với mỗi $v \in V$, $A(v)$ là tập các bộ (v, u, w) trong đó w là trọng số của cung (v, u)
  - $A(1) = \{(1, 6, 3), (1, 3, 7)\}$
  - $A(2) = \{(2, 4, 9), (2, 5, 6)\}$
  - $A(3) = \{(3, 4, 8)\}$
  - $A(4) = \{(4, 5, 2)\}$
  - $A(5) = \{\}$
  - $A(6) = \{(6, 3, 5), (6, 2, 4)\}$



DUYỆT ĐỒ THỊ

- Duyệt các đỉnh của đồ thị theo một thứ tự nào đó
- Các đỉnh được duyệt (thăm) đúng 1 lần
- Hai phương pháp cơ bản:
  - Duyệt theo chiều sâu (DFS)
  - Duyệt theo chiều rộng (BFS)

DUYỆT THEO CHIỀU SÂU

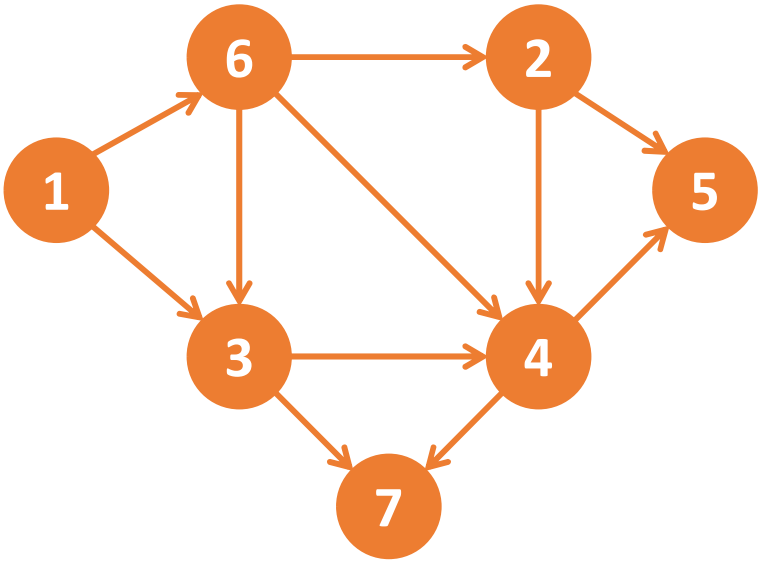
- DFS(*u*): duyệt theo chiều sâu bắt đầu từ đỉnh *u*
  - Nếu tồn tại đỉnh *v* trong danh sách kề của *u* chưa được thăm thì tiến hành thăm *v* và gọi DFS(*v*)
  - Nếu tất cả các đỉnh kề với *u* đã được thăm thì DFS quay trở lại đỉnh *x* mà từ đó thăm *u* để tiến hành thăm các đỉnh khác kề với *x* mà chưa được thăm. Lúc này đỉnh *u* được gọi là *đã duyệt xong*
- Cấu trúc dữ liệu: với mỗi đỉnh *v* của đồ thị
  - *p*(*v*): là đỉnh từ đó thăm *v*
  - *d*(*v*): thời điểm *v* được thăm nhưng chưa duyệt xong
  - *f*(*v*): thời điểm đỉnh *v* đã được duyệt xong
  - color(*v*)
    - WHITE: chưa thăm
    - GRAY: đã được thăm nhưng chưa duyệt xong
    - BLACK: đã duyệt xong

DUYỆT THEO CHIỀU SÂU

```
DFS(u) {
  t = t + 1;
  d(u) = t;
  color(u) = GRAY;
  foreach (đỉnh v kề với u) {
    if(color(v) = WHITE) {
      p(v) = u;
      DFS(v);
    }
  }
  t = t + 1;
  f(u) = t;
  color(u) = BLACK;
}
```

```
DFS() {
  foreach (đỉnh u thuộc V) {
    color(u) = WHITE;
    p(u) = NIL;
  }
  foreach(đỉnh u thuộc V) {
    if(color(u) = WHITE) {
      DFS(u);
    }
  }
}
```

DUYỆT THEO CHIỀU SÂU



đỉnh	1	2	3	4	5	6	7
d							
f							
p	-	-	-	-	-	-	-
color	W	W	W	W	W	W	W

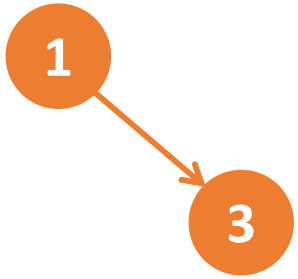
DUYỆT THEO CHIỀU SÂU

DFS(1)

đỉnh	1	2	3	4	5	6	7
d	1						
f							
p	-	-	-	-	-	-	-
color	G	W	W	W	W	W	W

DUYỆT THEO CHIỀU SÂU

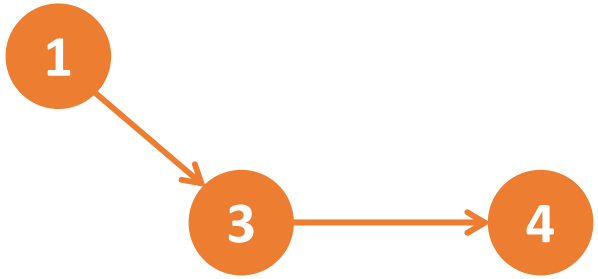
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2				
f							
p	-	-	1	-	-	-	-
color	G	W	G	W	W	W	W

DUYỆT THEO CHIỀU SÂU

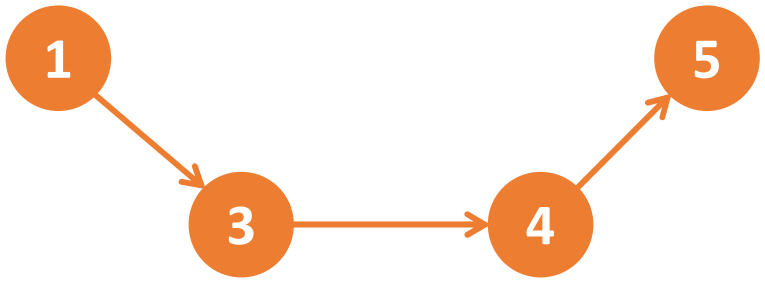
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3			
f							
p	-	-	1	3	-	-	-
color	G	W	G	G	W	W	W

DUYỆT THEO CHIỀU SÂU

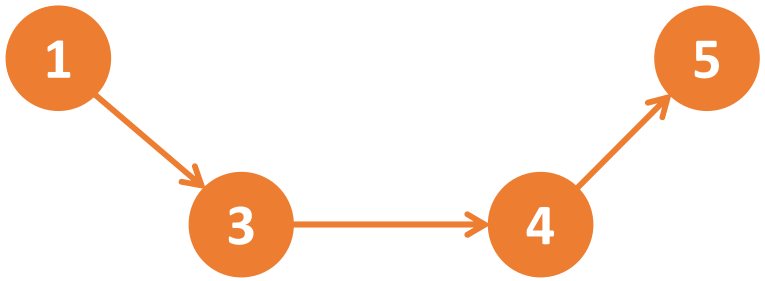
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		
f							
p	-	-	1	3	4	-	-
color	G	W	G	G	G	W	W

DUYỆT THEO CHIỀU SÂU

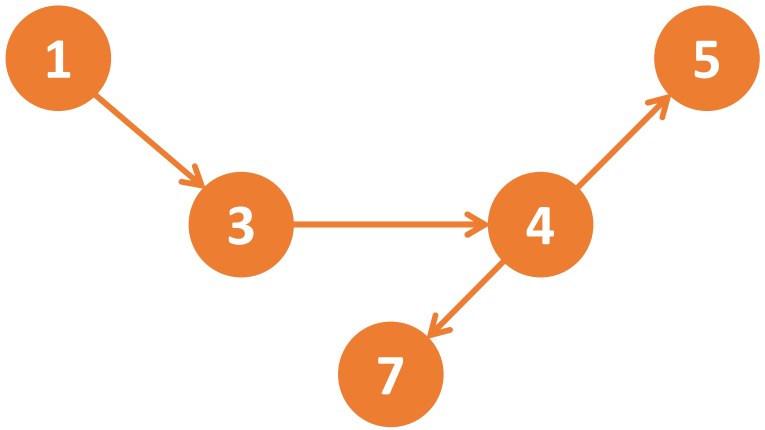
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		
f					5		
p	-	-	1	3	4	-	-
color	G	W	G	G	B	W	W

DUYỆT THEO CHIỀU SÂU

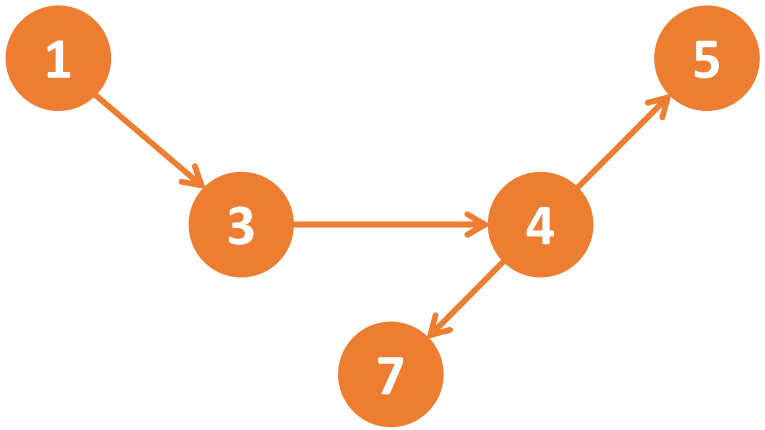
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		6
f					5		
p	-	-	1	3	4	-	4
color	G	W	G	G	B	W	G

DUYỆT THEO CHIỀU SÂU

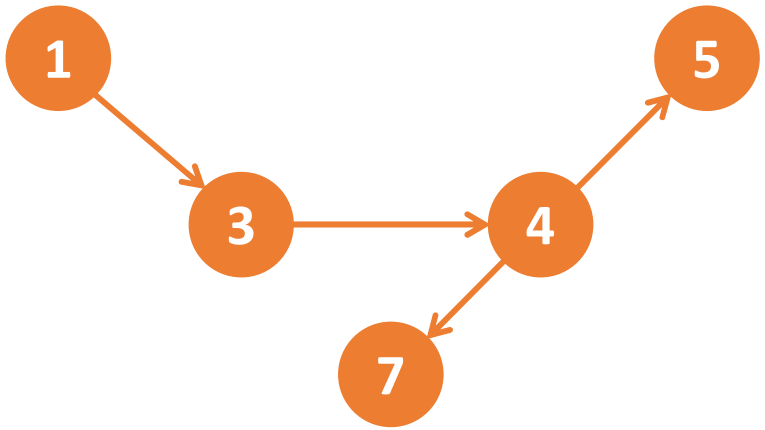
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		6
f					5		7
p	-	-	1	3	4	-	4
color	G	W	G	G	B	W	B

DUYỆT THEO CHIỀU SÂU

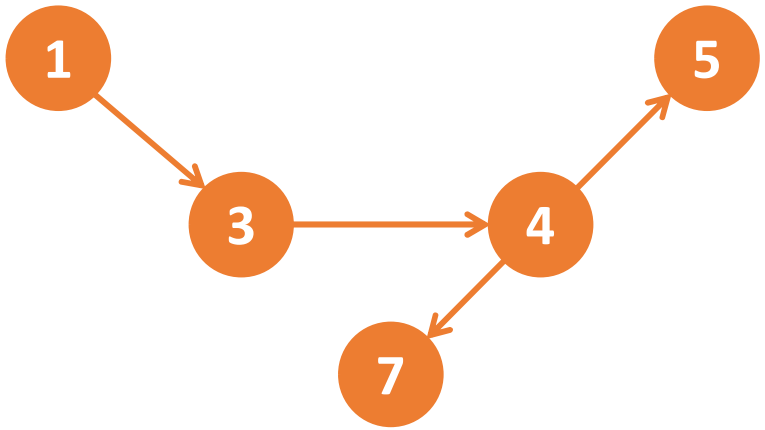
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		6
f				8	5		7
p	-	-	1	3	4	-	4
color	G	W	G	B	B	W	B

DUYỆT THEO CHIỀU SÂU

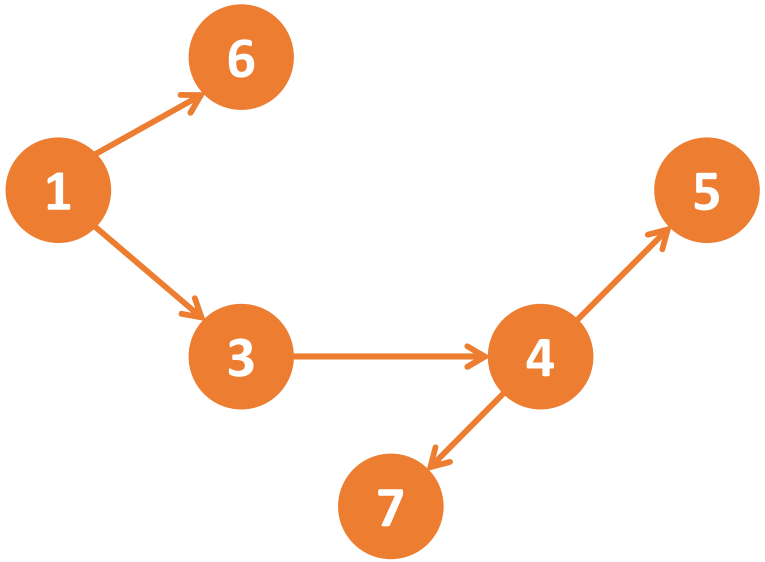
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		6
f			9	8	5		7
p	-	-	1	3	4	-	4
color	G	W	B	B	B	W	B

DUYỆT THEO CHIỀU SÂU

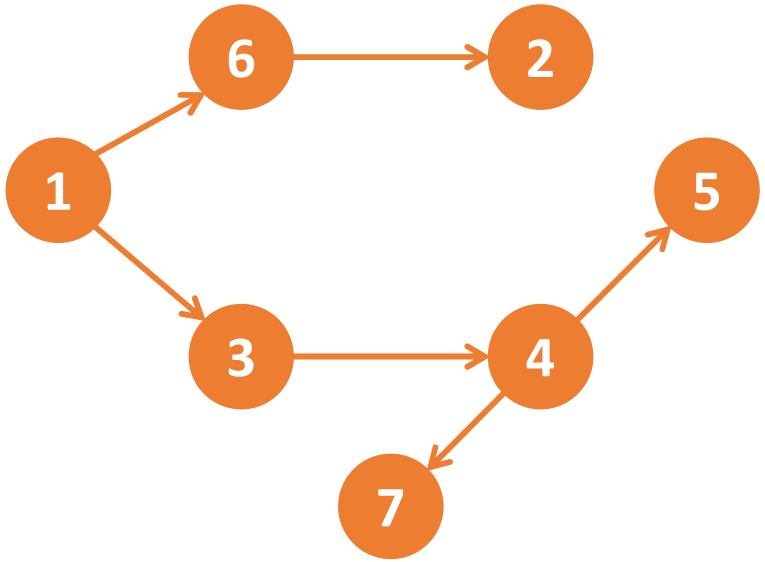
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4	10	6
f			9	8	5		7
p	-	-	1	3	4	1	4
color	G	W	B	B	B	G	B

DUYỆT THEO CHIỀU SÂU

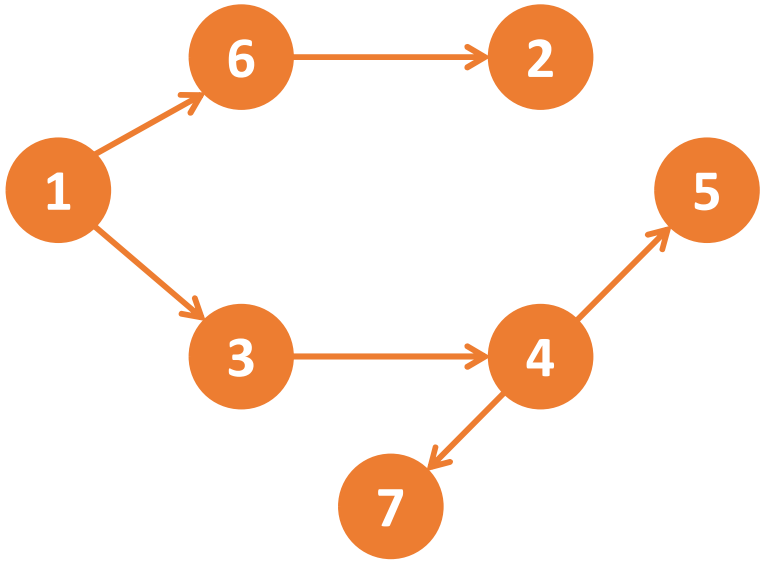
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f			9	8	5		7
p	-	6	1	3	4	1	4
color	G	G	B	B	B	G	B

DUYỆT THEO CHIỀU SÂU

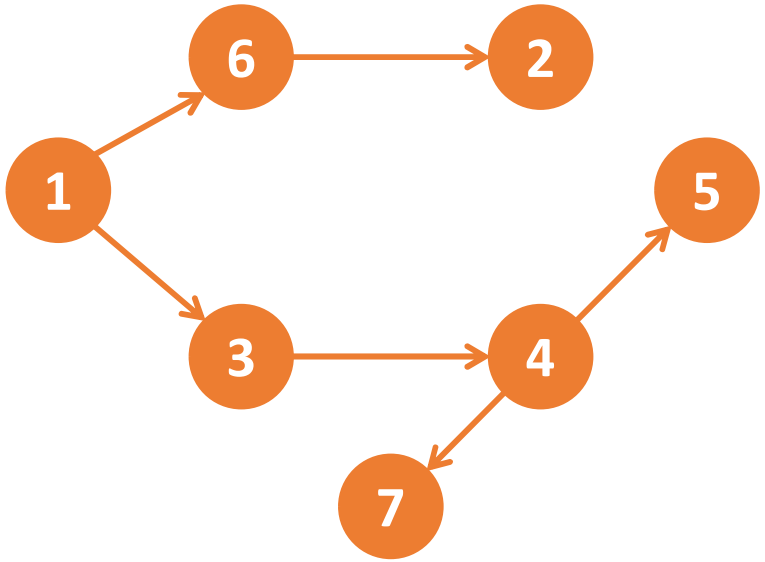
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f		12	9	8	5		7
p	-	6	1	3	4	1	4
color	G	B	B	B	B	G	B

DUYỆT THEO CHIỀU SÂU

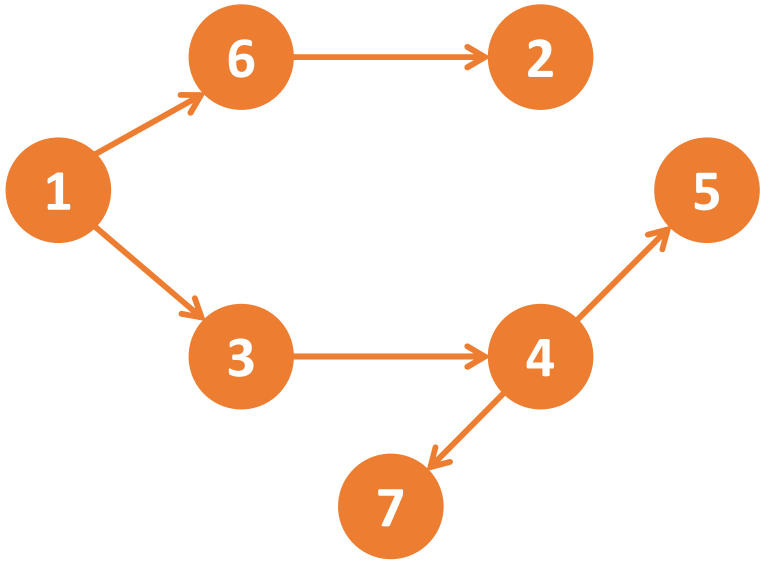
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f		12	9	8	5	13	7
p	-	6	1	3	4	1	4
color	G	B	B	B	B	B	B

DUYỆT THEO CHIỀU SÂU

DFS(1)



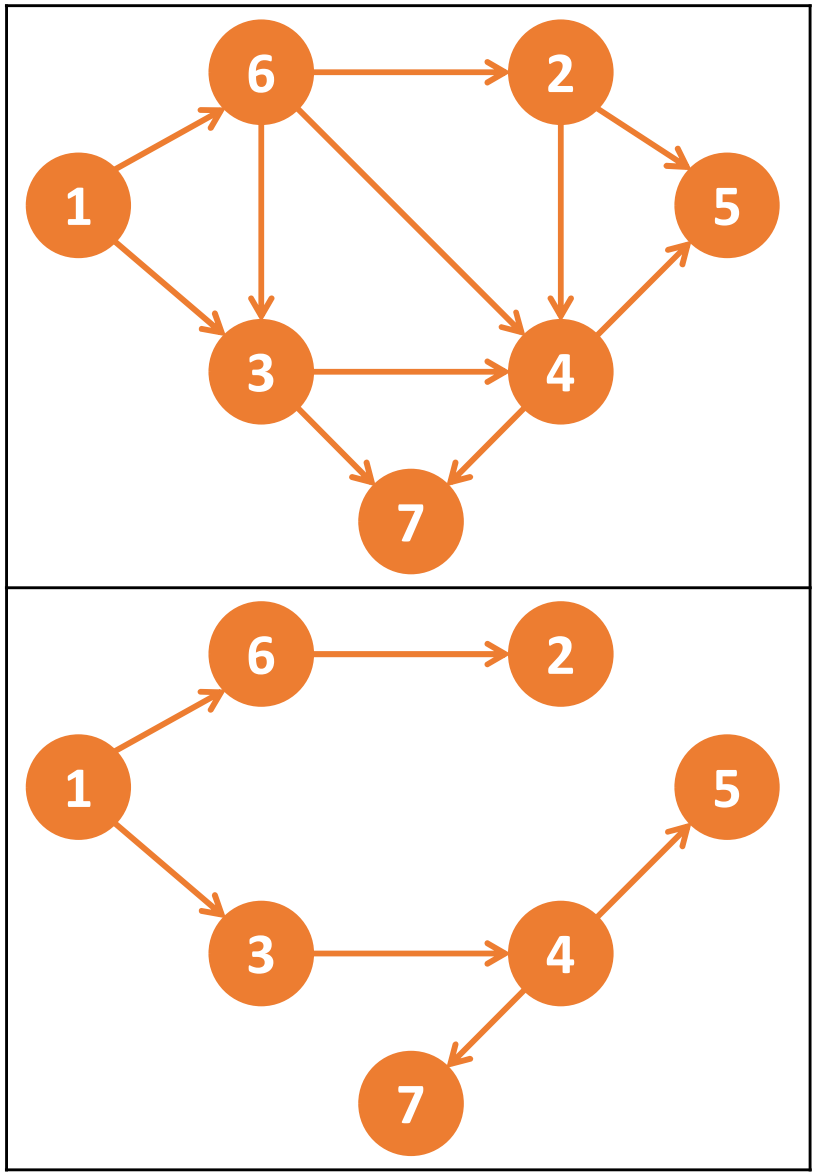
đỉnh	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f	14	12	9	8	5	13	7
p	-	6	1	3	4	1	4
color	B	B	B	B	B	B	B

DUYỆT THEO CHIỀU SÂU

- Kết quả DFS trên đồ thị sẽ cho 1 rừng, bao gồm các cây DFS
- Phân loại cạnh
  - Cạnh của cây (tree edge): (u,v) là cạnh của cây nếu v được thăm từ u
  - Cạnh ngược (back edge): (u,v) là cạnh ngược nếu v là tổ tiên của u trong cây DFS
  - Cạnh thuận (forward edge): (u,v) là cạnh thuận nếu u là tổ tiên của v trong cây DFS
  - Cạnh ngang (crossing edge): các cạnh còn lại

DUYỆT THEO CHIỀU SÂU

- Phân loại cạnh
  - Cạnh của cây: (1, 6), (1, 3), (6, 2), (3, 4), (4, 5), (4, 7)
  - Cạnh ngược:
  - Cạnh thuận: (3, 7)
  - Cạnh ngang: (6, 3), (6, 4), (2, 4), (2,5)



DUYỆT THEO CHIỀU SÂU

- Cài đặt thuật toán DFS tìm các thành phần liên thông của đồ thị

DUYỆT THEO CHIỀU SÂU

- Cài đặt thuật toán minh họa thuật toán DFS tìm các thành phần liên thông của đồ thị
- Dữ liệu đầu vào
  - Dòng 1: ghi n và m (số đỉnh và số cạnh của đồ thị)
  - Dòng i+1 (i = 1,…,m): ghi u và v là đầu mút của 1 cạnh (u,v) của đồ thị
- Kết quả đầu ra
  - Dòng thứ k: ghi danh sách các đỉnh thuộc thành phần liên thông thứ k

DUYỆT THEO CHIỀU SÂU

```
#include <stdio.h>
#define N  1000
int n,m;
int A[N][N];// A[v] is the list of adjacent nodes of v
int szA[N];// szA[v] is the size of A[v]
// data structure for connected component
int ncc;
int icc[N];// icc[u] is the  index of connected component of u
void input(){
    scanf("%d%d",&n,&m);
    for(int v = 1; v <= n; v++) szA[v] = 0;
    for(int k = 1; k <= m; k++){
        int u,v;
        scanf("%d%d",&u,&v);
        A[v][szA[v]] = u; A[u][szA[u]] = v;
        szA[v]++; szA[u]++;
    }
}
```

DUYỆT THEO CHIỀU SÂU

```
void dfs(int u){
    icc[u] = ncc;
    for(int i = 0; i < szA[u]; i++){
        int v = A[u][i];
        if(icc[v] == -1){// not visited
            dfs(v);
        }
    }
}
```

DUYỆT THEO CHIỀU SÂU

```
void dfsGraph(){
    for(int v = 1; v <= n; v++) icc[v] = -1;
    ncc = 0;
    for(int v = 1; v <= n; v++) if(icc[v] == -1){
        ncc++;
        dfs(v);
    }
    for(int k = 1; k <= ncc; k++){
        printf("connected component %d: ",k);
        for(int v = 1; v <= n; v++) if(icc[v] == k) printf("%d ",v);
        printf("\n");
    }
}
void main(){
    input();
    dfsGraph();
}
```

DUYỆT THEO CHIỀU RỘNG

- BFS(*u*): duyệt theo chiều rộng xuất phát từ đỉnh *u*
  - Thăm các đỉnh *u*
  - Thăm các đỉnh kề với u mà chưa được thăm (gọi là các đỉnh mức 1)
  - Thăm các đỉnh kề với các đỉnh mức 1 mà chưa được thăm (gọi là các đỉnh mức 2)
  - Thăm các đỉnh kề với các đỉnh mức 2 mà chưa được thăm (gọi là các đỉnh mức 3)
  - . . .
- Sử dụng cấu trúc hàng đợi (queue) để cài đặt

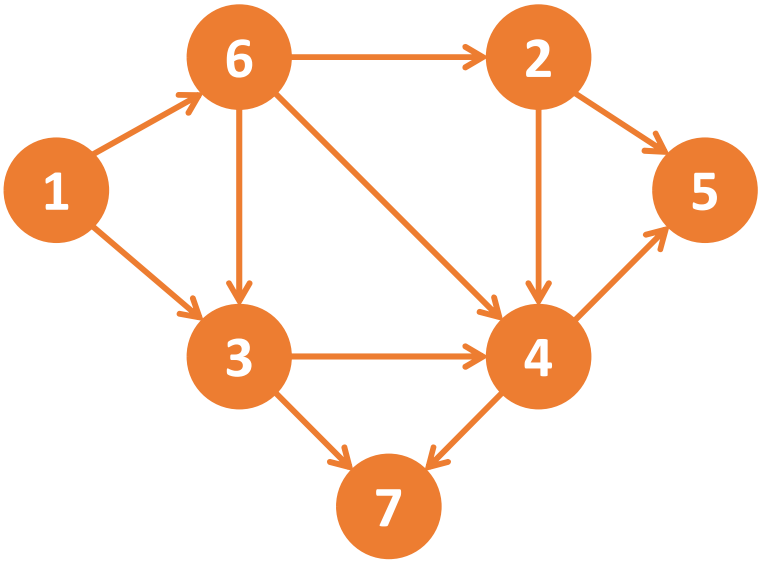
DUYỆT THEO CHIỀU RỘNG

```
BFS(u) {
  d(u) = 0;
  khởi tạo hàng đợi Q;
  enqueue(Q,u);
  color(u) = GRAY;
  while(Q khác rỗng) {
    v = dequeue(Q);
    foreach(x kề với v) {
      if(color(x) = WHITE){
        d(x) = d(v) + 1;
        color(x) = GRAY;
        enqueue(Q,x);
      }
    }
  }
}
```

```
BFS() {
  foreach (đỉnh u thuộc V) {
    color(u) = WHITE;
    p(u) = NIL;
  }
  foreach(đỉnh u thuộc V) {
    if(color(u) = WHITE) {
       BFS(u);
    }
  }
}
```

DUYỆT THEO CHIỀU RỘNG

BFS(1)



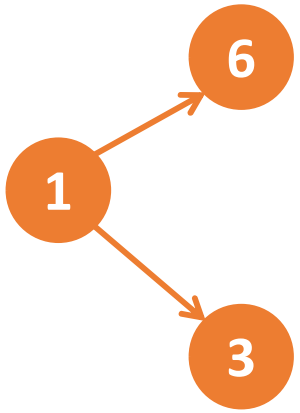
DUYỆT THEO CHIỀU RỘNG

BFS(1)

1

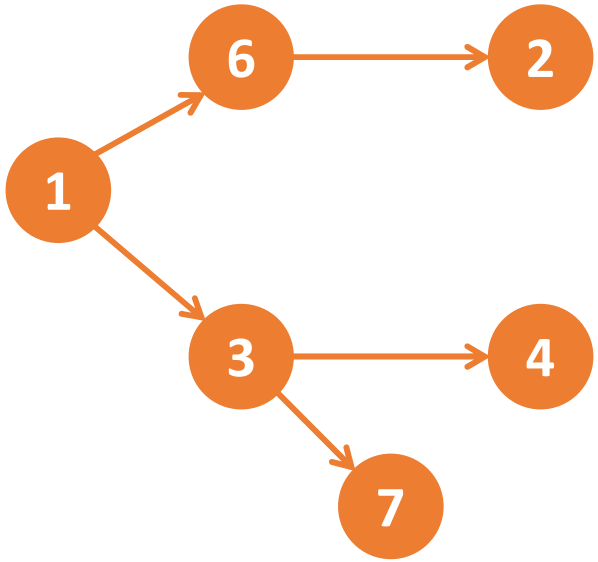
DUYỆT THEO CHIỀU RỘNG

BFS(1)



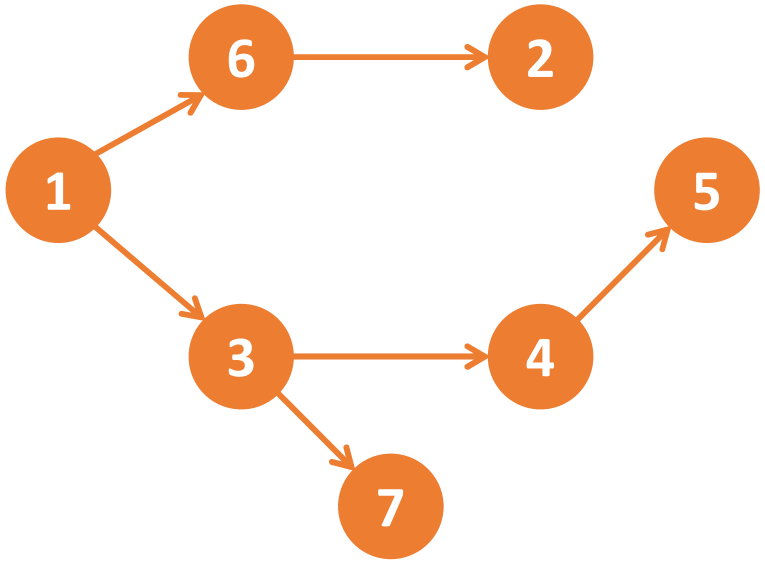
DUYỆT THEO CHIỀU RỘNG

BFS(1)



DUYỆT THEO CHIỀU RỘNG

BFS(1)



DUYỆT THEO CHIỀU RỘNG

- Cài đặt thuật toán minh họa thuật toán BFS tìm các thành phần liên thông của đồ thị
- Dữ liệu đầu vào
  - Dòng 1: ghi n và m (số đỉnh và số cạnh của đồ thị)
  - Dòng i+1 (i = 1,…,m): ghi u và v là đầu mút của 1 cạnh (u,v) của đồ thị
- Kết quả đầu ra
  - Dòng thứ k: ghi danh sách các đỉnh thuộc thành phần liên thông thứ k

DUYỆT THEO CHIỀU RỘNG

```
#include <stdio.h>
#define N 1000
// Data Structure for the queue
int q[N];
int head = -1, tail = -1;
// data structure for the graph
int n,m;// number of nodes and edges
int A[N][N]; // A[v] is the list of adjacent nodes to v
int szA[N];// szA[v] is the size of A[v]

// data structure for BFS connected component
int ncc; // number of connected component
int icc[N]; // icc[v] is the index of connected component of v
int d[N];// d[v] is the distance from the starting node to v in the BFS tree
```

DUYỆT THEO CHIỀU RỘNG

```
void initQueue(){
    head = -1; tail = -1;
}
void pushQ(int v){
    tail++; q[tail] = v; if(head == -1) head = tail;
}
int popQ(){
    if(head == -1) return -1;// queue empty
    int v = q[head];
    if(head == tail){
        head = -1; tail = -1;
    }else
        head++;
    return v;
}
int empty(){
    return head == -1 && tail == -1;
}
```

DUYỆT THEO CHIỀU RỘNG

```
void input(){
    scanf("%d%d",&n,&m);
    for(int v = 1; v <= n; v++) szA[v] = 0;
    for(int k = 1; k <= m; k++){
        int u,v;
        scanf("%d%d",&u,&v);
        A[v][szA[v]] = u; A[u][szA[u]] = v;
        szA[v]++; szA[u]++;
    }
}
```

DUYỆT THEO CHIỀU RỘNG

```
void bfs(int u){
    initQueue();
    icc[u] = ncc;
    pushQ(u);
    d[u] = 0;
    while(!empty()){
        int v = popQ();
        for(int i = 0; i < szA[v]; i++){
            int x = A[v][i];
            if(d[x] == -1){
                d[x] = d[v] + 1;
                pushQ(x);
                icc[x] = ncc;
            }
        }
    }
}
```

DUYỆT THEO CHIỀU RỘNG

```
void bfsGraph(){
    for(int v = 1; v <= n; v++) d[v] = -1;
    ncc = 0;
    for(int v = 1; v <= n; v++) if(d[v] == -1){
        ncc++;
        bfs(v);
    }
    for(int k = 1; k <= ncc; k++){
        printf("Connected component %d: ",k);
        for(int v = 1; v <= n; v++) if(icc[v] == k) printf("%d ",v);
        printf("\n");
    }
}
int main(){
    input();
    bfsGraph();
}
```

Cây khung nhỏ nhất của đồ thị

- Cho đồ thị vô hướng $G = (V, E, w)$.
  - Mỗi cạnh $(u, v) \in E$ có trọng số $w(u, v)$
  - Nếu $(u, v) \notin E$ thì $w(u, v) = \infty$
- Một cây khung của G là một đồ thị con liên thông của G không chứa chu trình và chứa tất cả các đỉnh của G.
  - $T = (V, F)$ trong đó $F \subseteq E$
  - Trọng số của T: $w(T) = \sum_{e \in F} w(e)$
- Tìm cây khung của G có trọng số nhỏ nhất

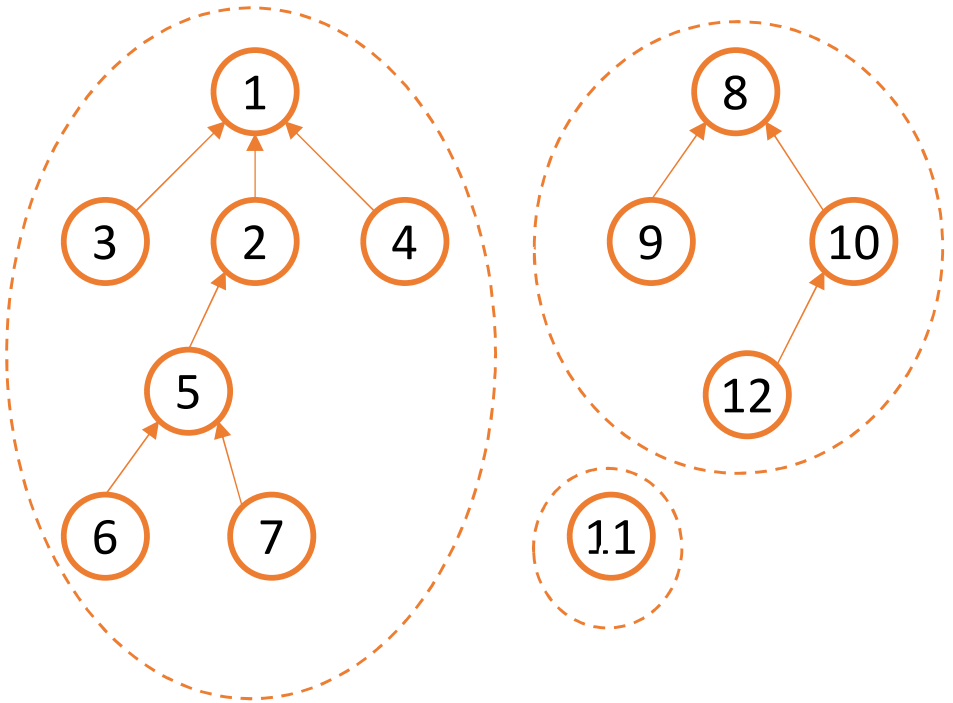
Cây khung nhỏ nhất: thuật toán Kruskal

- Thuật toán tham làm
  - Mỗi bước, chọn cạnh nhỏ nhất bổ sung vào T với điều kiện không tạo ra chu trình

```
KRUSKAL(G = (V,E)){
    ET = {}; C = E;
    while(|ET| < |V|-1 and |C| > 0){
        e = select minimum-cost edge of C;
        C = C \ {e};
        if(ET ∪ {e} create no cycle){
            ET = ET ∪ {e};
        }
    }
    if(|ET| = |V|-1) return ET;
    else return null;
}
```

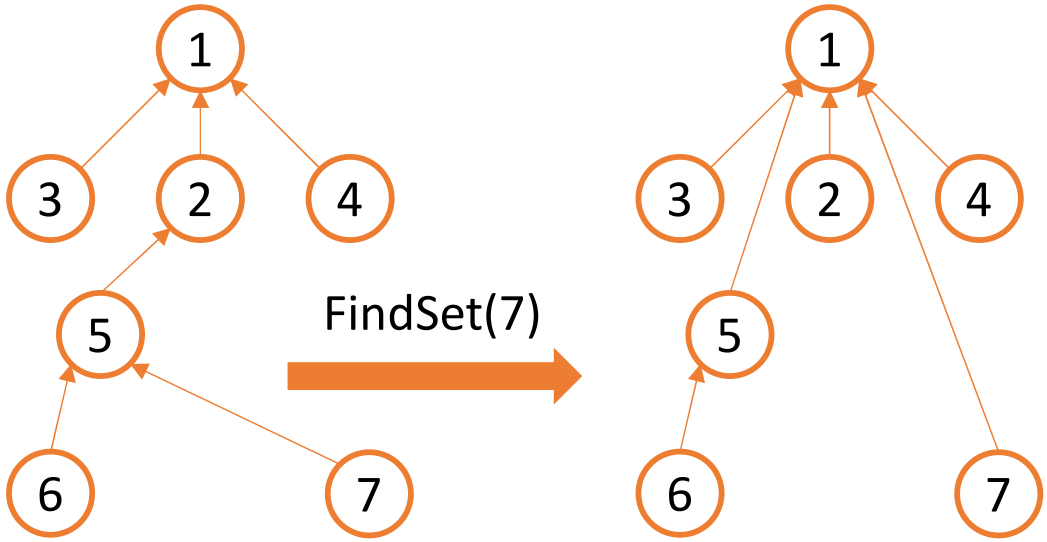
Disjoint Sets

- Là cấu trúc dữ liệu biểu diễn các tập không giao nhau với 2 thao tác chính
  - FindSet(x): trả về định danh của tập chứa x
  - Unify(r1, r2): Hợp nhất 2 tập hợp định danh là r1 và r2 làm một
- Mỗi tập được biểu diễn bởi cây có gốc
  - Mỗi nút của cây là một phần tử
  - Mỗi nút có 1 nút cha duy nhất (cha của nút gốc là chính nó)
  - Nút gốc là định danh của tập



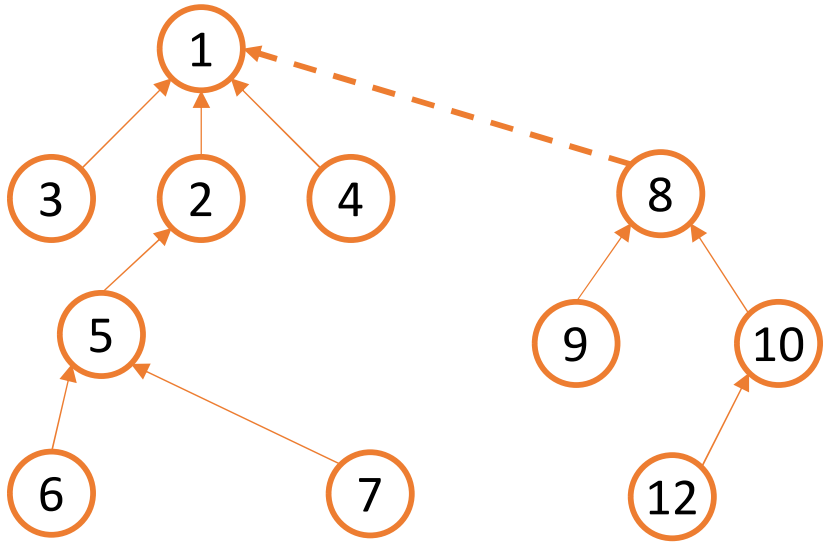
Disjoint Sets

- Thao tác FindSet(x)
  - Di chuyển từ x theo đường đi duy nhất từ nó đến nút gốc của cây chứa x (thời gian tính tỉ lệ với độ dài của đường đi)
  - Điều chỉnh cây mỗi khi thực hiện truy vấn: biến tất cả các nút dọc theo đường đi từ x đến nút gốc thành nút con trực tiếp của nút gốc → Rút ngắn thời gian của các truy vấn về sau



Disjoint Sets

- Thao tác Unify(r1, r2)
  - Biến một trong hai nút r1, hoặc r2 làm nút con của nút còn lại (tiêu chí nút nào có độ cao thấp hơn thì biến thành nút con của nút còn lại để đảm bảo duy trì độ cao của các cây thấp)



- Việc cài đặt duy trì 2 cấu trúc
  - p(x): cha của nút x
  - r(x): hạng của nút x (độ cao)

Độ cao của 8 nhỏ hơn độ cao của 1, nên biến 8 thành con của 1

Cây khung nhỏ nhất: thuật toán Kruskal

```
#include <iostream>
#define MAX 100001

using namespace std;

// data structure for input graph
int N, M;
int u[MAX];
int v[MAX];
int c[MAX];
int ET[MAX];
int nET;
// data structure for disjoint-set
int r[MAX];// r[v] is the rank of the set v
int p[MAX];// p[v] is the parent of v
long long rs;
```

Cây khung nhỏ nhất: thuật toán Kruskal

```
void unify(int x, int y){
    if(r[x] > r[y]) p[y] = x;
    else{
        p[x] = y;
        if(r[x] == r[y]) r[y] = r[y] + 1;
    }
}
void makeSet(int x){
    p[x] = x;
    r[x] = 0;
}
int findSet(int x){
    if(x != p[x])
        p[x] = findSet(p[x]);
    return p[x];
}
```

Cây khung nhỏ nhất: thuật toán Kruskal

```
void swap(int& a, int& b){
    int tmp = a; a = b; b = tmp;
}
void swapEdge(int i, int j){
    swap(c[i],c[j]);    swap(u[i],u[j]);    swap(v[i],v[j]);
}
int partition(int L, int R, int index){
    int pivot = c[index];
    swapEdge(index,R);
    int storeIndex = L;
    for(int i = L; i <= R-1; i++){
        if(c[i] < pivot){
            swapEdge(storeIndex,i);
            storeIndex++;
        }
    }
    swapEdge(storeIndex,R);
    return storeIndex;
}
```

Cây khung nhỏ nhất: thuật toán Kruskal

```
void quickSort(int L, int R){
    if(L < R){
        int index = (L+R)/2;
        index = partition(L,R,index);
        if(L < index) quickSort(L,index-1);
        if(index < R) quickSort(index+1,R);
    }
}
void quickSort(){
    quickSort(0,M-1);
}
```

Cây khung nhỏ nhất: thuật toán Kruskal

```
void solve(){
    for(int x = 1; x <= N; x++)  makeSet(x);
    quickSort();
    rs = 0;
    int count = 0;
    nET = 0;
    for(int i = 0;i < M; i++){
        int ru = findSet(u[i]);
        int rv = findSet(v[i]);
        if(ru != rv){
            unify(ru,rv);
            nET++;  ET[nET] = i;
            rs += c[i];
            count++;
            if(count == N-1) break;
        }
    }
    cout << rs;
}
```

Cây khung nhỏ nhất: thuật toán Kruskal

```
void input(){
    cin >> N >> M;
    for(int i = 0; i < M; i++){
        cin >> u[i] >> v[i] >> c[i];
    }
}
int main(){
    input();
    solve();
 }
```

Đường đi ngắn nhất – thuật toán Dijkstra

- Cho đồ thị trọng số không âm $G = (V, E, w)$.
  - Mỗi cạnh $(u, v) \in E$ có trọng số không âm $w(u, v)$
  - Nếu $(u, v) \notin E$ thì $w(u, v) = \infty$
- Cho đỉnh s của V, thì đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại của G

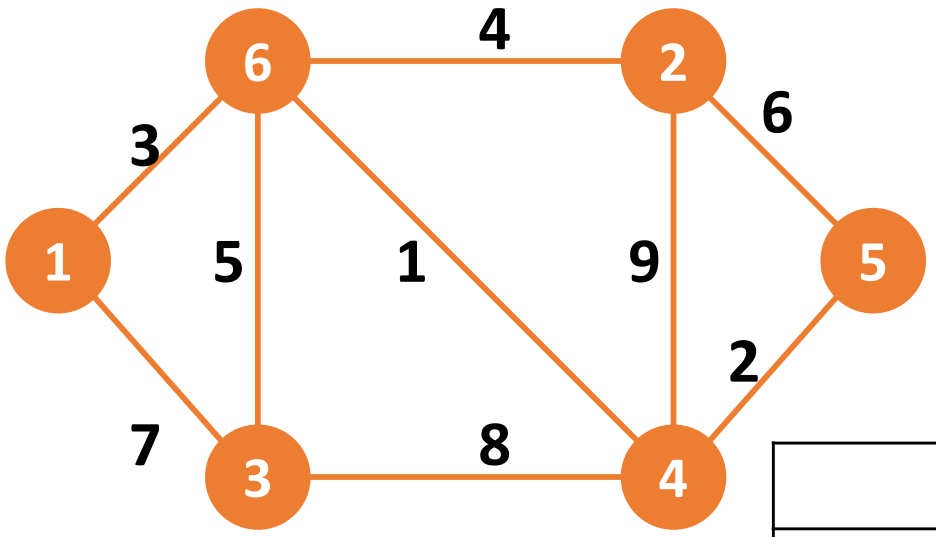
Đường đi ngắn nhất – thuật toán Dijkstra

- Thuật toán Dijkstra:
  - Mỗi đỉnh $v \in V$:
    - **P**(v) đường đi cận trên của đường đi ngắn nhất từ s đến v
    - $d(v)$: trọng số của **P**(v)
    - $p(v)$: đỉnh trước đỉnh v trên **P**(v)
  - Khởi tạo
    - **P**(v) = $\langle s, v \rangle$, $d(v) = w(s,v)$, $p(v) = s$
  - Cải tiến cận trên
    - Nếu phát hiện một đỉnh u sao cho $d(v) > d(u) + w(u, v)$ thì cập nhật:
      - $d(v) = d(u) + w(u, v)$
      - $p(v) = u$

Đường đi ngắn nhất – thuật toán Dijkstra

```
Dijkstra(G = (V,E,w)) {
  for(v ∈ V) {
     d(v) = w(s,v); p(v) = s;
  }
  S = V \ {s};
  while(S ≠ {}) {
    u = select a node ∈ S having minimum d(u);
    S = S \ {u};
    for(v ∈ S) {
      if(d(v) > d(u) + w(u,v) {
        d(v) = d(u) + w(u,v);  p(v) = u;
      }
    }
  }
}
```

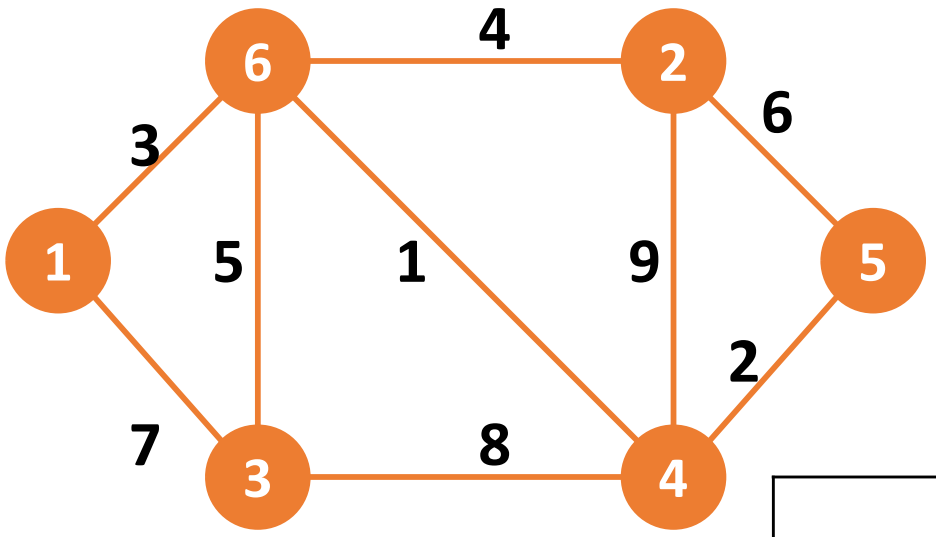
Đường đi ngắn nhất – thuật toán Dijkstra



- Mỗi ô của bảng tương ứng với 1 đỉnh v của bảng có nhãn $(d(v), p(v))$
- Nút nguồn $s = 1$

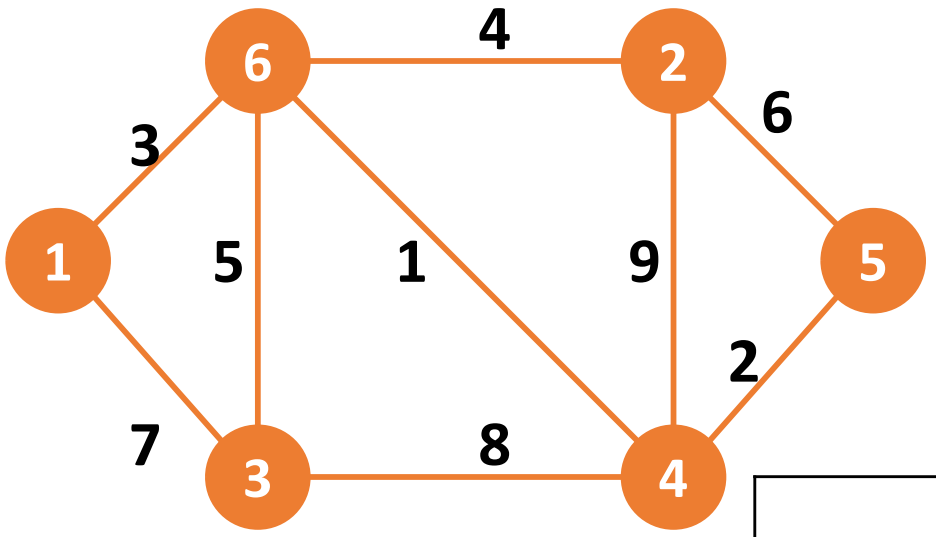
	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1)
Step 1	-					
Step 2	-					
Step 3	-					
Step 4	-					
Step 5	-					

Đường đi ngắn nhất – thuật toán Dijkstra



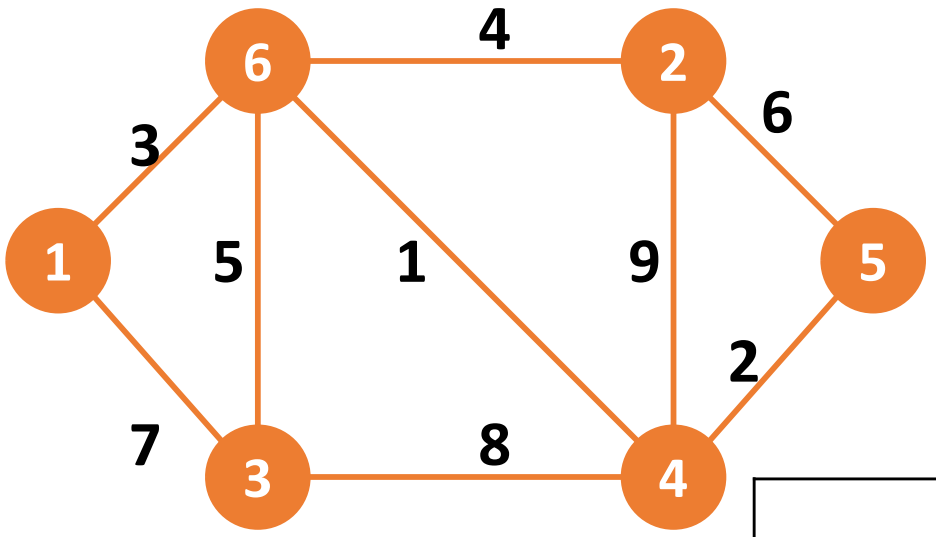
	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Step 1	-	(7,6)	(7,1)	(4,6)	(∞,1)	-
Step 2	-					-
Step 3	-					-
Step 4	-					-
Step 5	-					-

Đường đi ngắn nhất – thuật toán Dijkstra



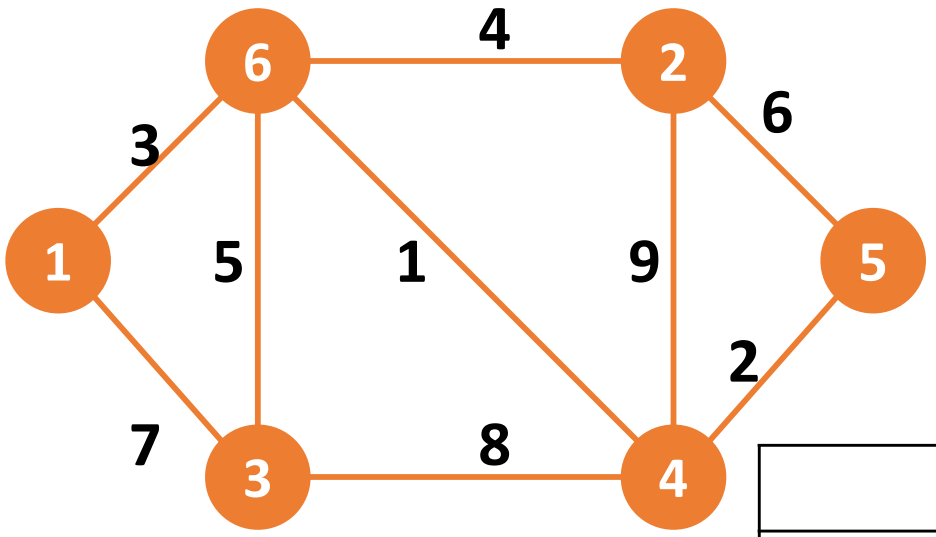
	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Step 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Step 2	-	(7,6)	(7,1)	-	(6, 4)	-
Step 3	-			-		-
Step 4	-			-		-
Step 5	-			-		-

Đường đi ngắn nhất – thuật toán Dijkstra



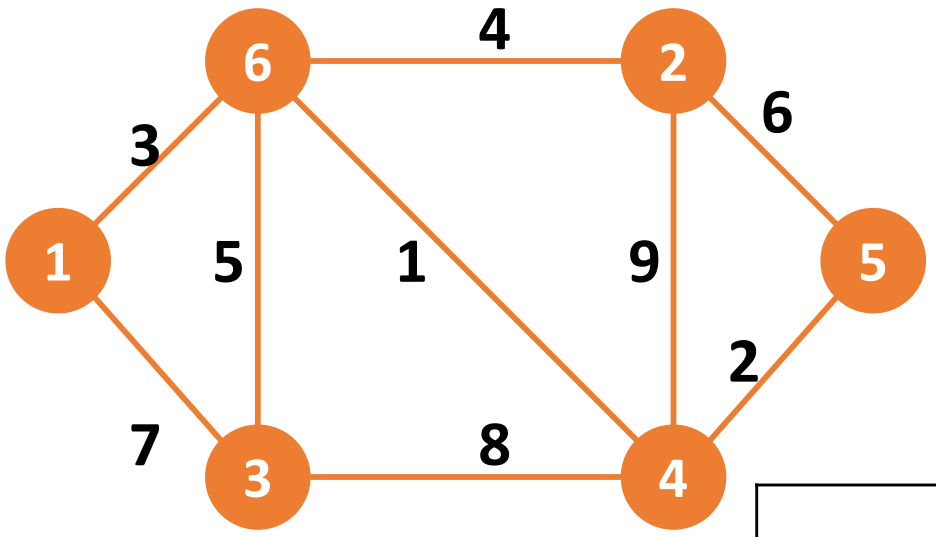
	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Step 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Step 2	-	(7,6)	(7,1)	-	(6, 4) *	-
Step 3	-	(7,6)	(7,1)	-	-	-
Step 4	-			-	-	-
Step 5	-			-	-	-

Đường đi ngắn nhất – thuật toán Dijkstra



	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Step 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Step 2	-	(7,6)	(7,1)	-	(6, 4) *	-
Step 3	-	(7,6)	(7,1) *	-	-	-
Step 4	-	(7,6)	-	-	-	-
Step 5	-		-	-	-	-

Đường đi ngắn nhất – thuật toán Dijkstra

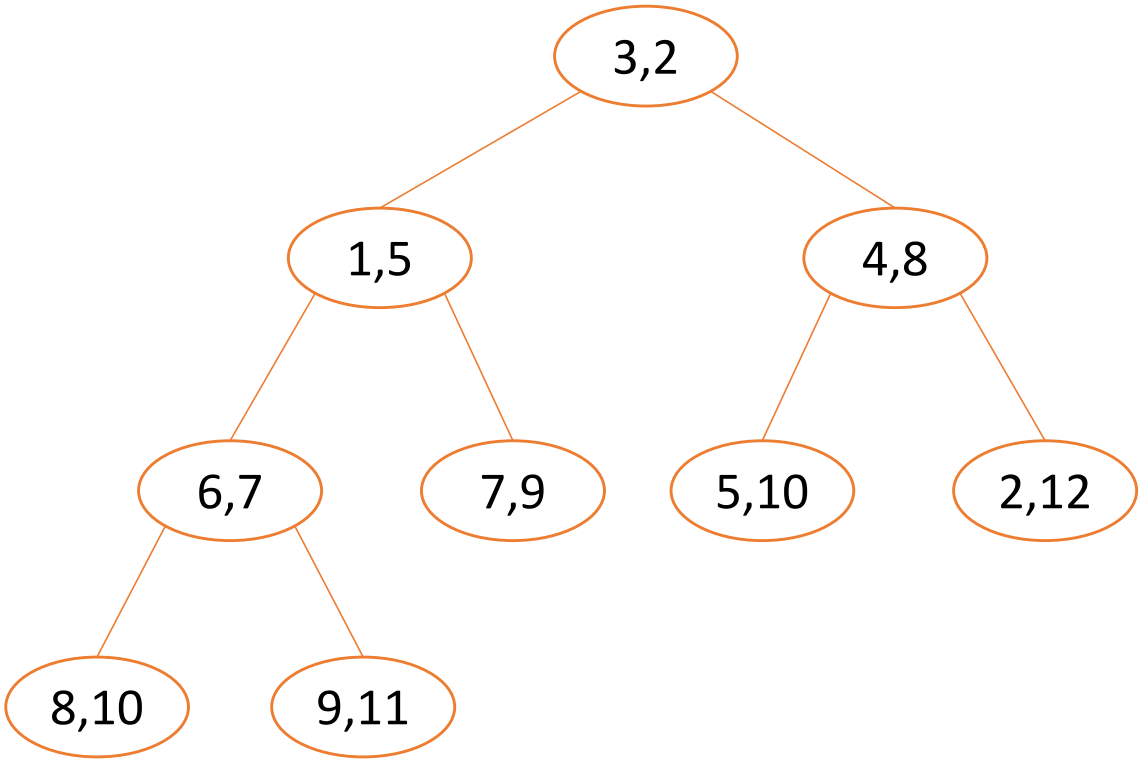


	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Step 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Step 2	-	(7,6)	(7,1)	-	(6, 4) *	-
Step 3	-	(7,6)	(7,1) *	-	-	-
Step 4	-	(7,6) *	-	-	-	-
Step 5	-	-	-	-	-	-

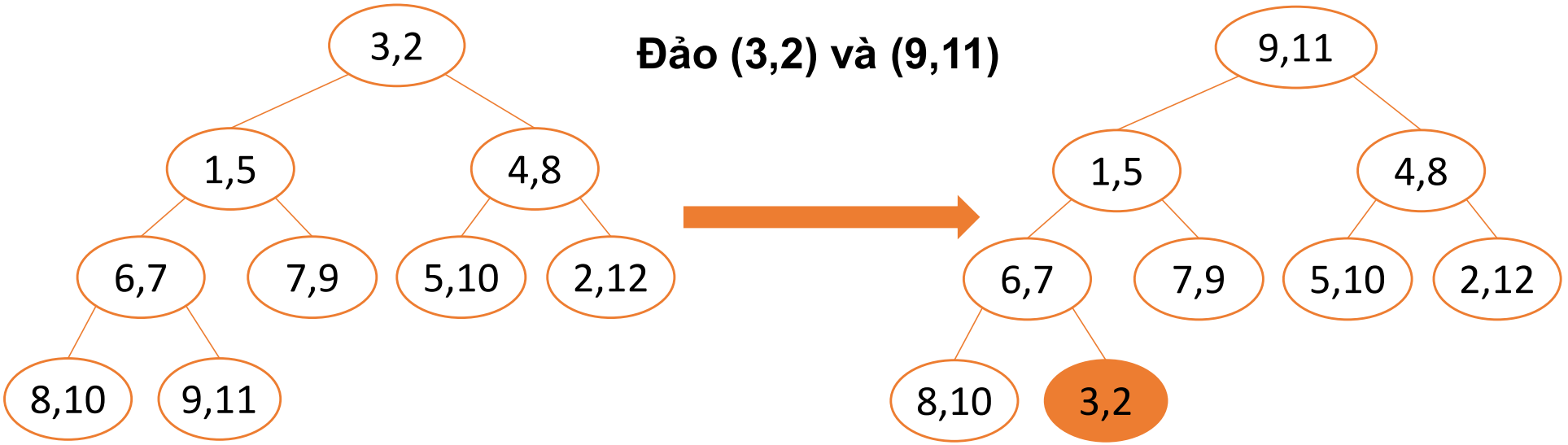
Đường đi ngắn nhất – thuật toán Dijkstra

- **Hàng đợi ưu tiên (Priority queues)**
  - Cấu trúc dữ liệu lưu trữ các đỉnh v và khóa d(v)
  - Các thao tác
    - (e,k) = deleteMin(): Lấy ra đỉnh e có khóa k nhỏ nhất
    - insert(v,k): chèn một nút e và khóa k của nó vào hàng đợi
    - updateKey(v,k): Cập nhật nút v với khóa mới bằng k
  - Cài đặt priority queue bằng cấu trúc min-heap
    - Các phần tử lưu trữ theo cây nhị phân đầy đủ
    - Khóa của mỗi nút (đỉnh của đồ thị) nhỏ hơn hoặc bằng khóa của 2 nút con

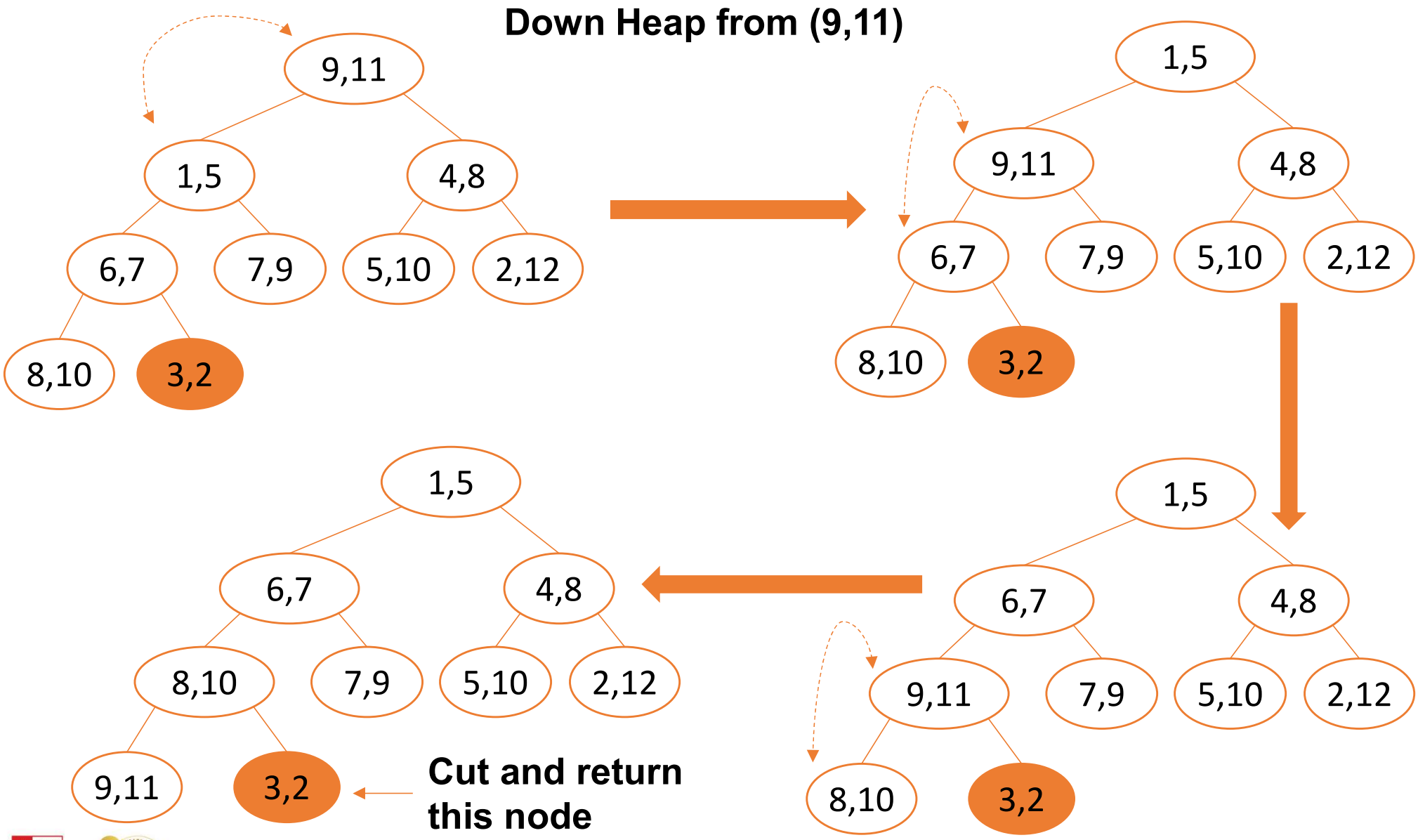
Đường đi ngắn nhất – thuật toán Dijkstra



Đường đi ngắn nhất – thuật toán Dijkstra



Đường đi ngắn nhất – thuật toán Dijkstra



Đường đi ngắn nhất – thuật toán Dijkstra

```
#include <stdio.h>
#include <vector>
#define MAX 100001
#define INF 1000000
using namespace std;

vector<int> A[MAX]; // A[v][i] is the i^th adjacent node to v
vector<int> c[MAX]; // c[v][i] is the weight of the i^th adjacent arc
                    // (v,A[v][i]) to v
int n,m; // number of nodes and arcs of the given graph
int s,t; // source and destination nodes

// priority queue data structure (BINARY HEAP)
int d[MAX];// d[v] is the upper bound of the length of the shortest path from s
           // to v (key)
int node[MAX];// node[i] the i^th element in the HEAP
int idx[MAX];// idx[v] is the index of v in the HEAP (idx[node[i]] = i)
int sH;// size of the HEAP
bool fixed[MAX];
```

Đường đi ngắn nhất – thuật toán Dijkstra

```
void swap(int i, int j){
    int tmp = node[i]; node[i] = node[j]; node[j] = tmp;
    idx[node[i]] = i; idx[node[j]] = j;
}
void upHeap(int i){
    if(i == 0) return;
    while(i > 0){
        int pi = (i-1)/2;
        if(d[node[i]] < d[node[pi]]){
            swap(i,pi);
        }else{
            break;
        }
        i = pi;
    }
}
```

Đường đi ngắn nhất – thuật toán Dijkstra

```
void downHeap(int i){
    int L = 2*i+1;
    int R = 2*i+2;
    int maxIdx = i;
    if(L < sH && d[node[L]] < d[node[maxIdx]]) maxIdx = L;
    if(R < sH && d[node[R]] < d[node[maxIdx]]) maxIdx = R;
    if(maxIdx != i){
        swap(i,maxIdx); downHeap(maxIdx);
    }
}
void insert(int v, int k){
    // add element key = k, value = v into HEAP
    d[v] = k;
    node[sH] = v;
    idx[node[sH]] = sH;
    upHeap(sH);
    sH++;
}
```

Đường đi ngắn nhất – thuật toán Dijkstra

```
int inHeap(int v){
    return idx[v] >= 0;
}
void updateKey(int v, int k){
    if(d[v] > k){
        d[v] = k;
        upHeap(idx[v]);
    }else{
        d[v] = k;
        downHeap(idx[v]);
    }
}
```

Đường đi ngắn nhất – thuật toán Dijkstra

```
int deleteMin(){
    int sel_node = node[0];
    swap(0,sH-1);
    sH--;
    downHeap(0);
    return sel_node;
}
```

Đường đi ngắn nhất – thuật toán Dijkstra

```
void input(){
    scanf("%d%d",&n,&m);
    for(int k = 1; k <= m; k++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        A[u].push_back(v);
        c[u].push_back(w);

    }
    scanf("%d%d",&s,&t);
}
```

Đường đi ngắn nhất – thuật toán Dijkstra

```
void init(int s){
    sH = 0;
    for(int v = 1; v <= n; v++){
        fixed[v] = false;  idx[v] = -1;
    }
    d[s] = 0;    fixed[s] = true;
    for(int i = 0; i < A[s].size(); i++){
        int v = A[s][i];
        insert(v,c[s][i]);
    }
}
```

Đường đi ngắn nhất – thuật toán Dijkstra

```
void solve(){
    init(s);
    while(sH > 0){
        int u = deleteMin(); fixed[u] = true;
        for(int i = 0; i < A[u].size(); i++){
            int v = A[u][i];
            if(fixed[v]) continue;
            if(!inHeap(v)){
                    int w = d[u] + c[u][i]; insert(v,w);
            }else{
                if(d[v] > d[u] + c[u][i]) updateKey(v,d[u]+c[u][i]);

            }
        }
    }
    int rs = d[t]; if(!fixed[t]) rs = -1;
    printf("%d",rs);
}
```

Đường đi ngắn nhất – thuật toán Dijkstra

```
int main(){
    input();
    solve();
}
```

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict